Scheduling on a machine with varying speed: Minimizing cost and energy via dual schedules

Nicole Megow* José Verschae†

November 28, 2012

Abstract

We study two types of problems related with scheduling on a machine of varying speed. In a static model, the speed function is (through an oracle) part of the input and we ask for a cost-efficient scheduling solution. In a dynamic model, deciding upon the speed is part of the scheduling problem and we are interested in the tradeoff between scheduling cost and speed-scaling cost, that is energy consumption. Such problems are relevant in production planning, project management, and in power-management of modern microprocessors.

We consider scheduling to minimize the total weighted completion time. As our main result, we present a PTAS for the static and the dynamic problem of scheduling on a single machine of varying speed. As a key to our results, we re-interprete our problem within the folkloric *two-dimensional Gantt chart*: instead of the standard approach of scheduling in the *time-dimension*, we construct scheduling solutions in the *weight-dimension*. We also give complexity results, more efficient algorithms for special cases, and a simple $(2+\varepsilon)$-approximation for preemptive dynamic speed-scaling with release dates. Our results also apply to the closely related problem of scheduling to minimize generalized global cost functions.

Key words: scheduling, speed-scaling, power-management, generalized cost functions, non-availability periods, energy-aware

1 Introduction

In several computation and production environments we face scheduling problems in which the speed of resources may vary. We distinguish mainly two types of varying speed scenarios: one, in which the speed is static and cannot be influenced, and another dynamic setting in which deciding upon the processor speed is part of the scheduling problem. The static setting occurs, e.g., in production environments where the speed of a resource may change due to overloading, aging, or in an extreme case it may be completely unavailable due to maintenance or failure. The dynamic setting finds application particularly in modern computer architectures, where speed-scaling is an important tool for power-management. Here we are interested in the tradeoff between the power consumption and the quality-of-services. Both research directions—scheduling on a machine with given speed fluctuation as well as scheduling including speed-scaling—have been pursued quite extensively, but seemingly separately from each other.

In this paper, we consider scheduling to minimize a standard measure for quality-of-service, the sum of weighted completion times $\sum_j w_j C_j$, on a machine of varying speed. For both settings, static and dynamic speed-scaling, we derive structural results, complexity results, and best possible (fully) polynomial time approximation schemes. Very useful in our arguments is the

*Department of Mathematics, Technische Universität Berlin, Germany. Email: `nmegow@math.tu-berlin.de`.
Supported by the German Science Foundation (DFG) under contract ME 3825/1.

†Departamento de Ingeniería Industrial, Universidad de Chile, Santiago, Chile. Email: `jverscha@ing.uchile.cl`.
Supported by the Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F.

geometric view of the min-sum scheduling problem in a *two-dimensional Gantt chart*. Crucial to our results is the deviation from the standard view of scheduling as a decision making process in the *time dimension* and switching to scheduling in the *weight dimension*, instead. This dual view allows us to cope with the highly sensitive speed changes in the time dimension which prohibit standard rounding, guessing, and approximation techniques.

Previous work

Research on scheduling on a machine of given varying speed has been mainly focused on the special case of scheduling with non-availability periods, see e.g. [9,16,17,20]. For $\min \sum w_j C_j$, only recently the first constant approximation was derived in [10]. In fact, their $(4+\varepsilon)$-approximation computes a universal sequence which has the same guarantee for any (unknown) speed function. If the speed is only increasing, there is known an efficient PTAS [21]. In this case the complexity remains an open question, whereas for general speed functions the problem is strongly NP-hard, even when for each job the weight and processing time are equal [22].

The problem of scheduling on a machine of varying speed is equivalent to scheduling on an ideal machine (of constant speed) but minimizing a more general cost function, $\sum w_j f(C_j)$, where f is a nondecreasing function. This identification is done via a straight forward transformation, where $f(C)$ denotes the time that the varying speed machine needs to process a work volume of C [12]. In particular, the special case of only nondecreasing (nonincreasing) speed functions corresponds to concave (convex) global cost functions. Recently, in [12] were given tight guarantees for the well-known Smith rule for all convex and all concave functions f. In this work is also shown that the problem for increasing piecewise linear cost functions is strongly NP-hard even for functions with only two slopes, and so is our problem when the machine speed takes only two distinct values.

Even more general min-sum cost functions have been studied, where each job may have its individual nondecreasing cost function. A $(2+\varepsilon)$-approximation was recently derived in [8]. For the more complex setting with release dates there is a randomized $\mathcal{O}(\log\log(n \max_j p_j))$-approximation known [4]. Clearly, these results translate also to the setting with varying machine speed.

Scheduling with dynamic speed-scaling was initiated in [23] and became a very active research field in the past fifteen years. Most work focuses on scheduling problems where jobs have deadlines by which they must finish. We refer to [2, 13] for an overview. The line of research closest to our work seems to be the one initiated by Pruhs et al. [19], and later continued by many others; see, e. g., [3, 5, 7] and the references therein. Most of these works study online algorithms to minimize total (or weighted) flow time plus energy. The closest result to ours seems to be a $\mathcal{O}(1)$-approximation algorithm for minimizing weighted flow time if the energy is controlled by a budget [5].

Our results

We give several best possible algorithms for problem variants that involve scheduling to minimize the total weighted completion time on a single machine that may vary its speed.

Our main result is an efficient PTAS (Section 3) for scheduling to minimize $\sum w_j C_j$ on a machine of varying speed (given by an oracle). This is best possible since the problem is strongly NP-hard, even when the machine speed takes only two distinct values [12]. We also provide an FPTAS (Section 5.3) for the case that there is a constant number of time intervals with different uniform speeds (and the max ratio of speeds is bounded). Our results generalize recent previous results such as a PTAS on a machine with only increasing speeds [21] and FPTASes for one non-availability period [14, 15].

Our results cannot be obtained with standard scheduling techniques which heavily rely on rounding processing requirements or completion times. Such approaches typically fail on machines that may change their speed since the slightest error introduced by rounding might provoke an unbounded increase in the solution cost. Similarly, adding any amount of idle time to the machine might be fatal. Our techniques completely avoid this difficulty by a change of paradigm.

To explain our ideas it is helpful to use a 2D-Gantt chart, see Section 2. As observed before, e.g., in [11], we obtain a *dual* scheduling problem by looking at the y-axis in a 2D-Gantt chart and switching the roles of the processing times and weights. In other words, a dual solution describes a schedule by specifying the remaining weight of the system at the moment a job completes. This simple idea avoids the difficulties on the time-axis and allows to combine old with new techniques for scheduling on the weight-axis.

In Section 4 we study the problem in which the machine can take an arbitrary non-negative speed (speed-scaling with a continuous spectrum of available speeds). For this case we show the rather surprising fact that the order of jobs in an optimal schedule is independent of the available energy. This follows by analyzing a convex program that models the optimal energy assignment for a given job permutation. We show that computing this universal optimal sequence corresponds to the problem of scheduling with a particular concave global cost function, which can be solved with our PTAS mentioned above, or with a PTAS for non-decreasing speed [21]. Interestingly, this reduction relies again on a problem transformation from time-space to weight-space in the 2D-Gantt chart. For a given scheduling sequence, we give an explicit formulae for computing the optimal energy (speed) assignment. Thus, we have a PTAS for speed-scaling and scheduling for a given energy budget. This, together with the fact that the optimal permutation is independent of the energy available, yields a $(1+\varepsilon)$-approximation of the Pareto curve which describes the near-optimal cost as a function of energy.

In many applications, including most modern computer architectures, machines are only capable of using a given number of discrete power (speed) states. We also provide in Section 5 an efficient PTAS for this more complex scenario. This algorithm is again based on our techniques using dual schedules. Furthermore, we obtain a $(1+\varepsilon)$-approximation of the Pareto frontier for the energy-cost bicriteria problem. On the other hand, we show that this problem is NP-hard even when there are only two speed states. We complement this result by giving an FPTAS for a constant number of available speeds.

It is not too hard to see that the (F)PTAS solution leads to a simple $(2+\varepsilon)$-approximation for the more general setting with release dates by applying preemptive list scheduling according to this order without changing the speed assignment.

We remark that all our results for the setting with given speed translate directly to a corresponding result for the equivalent problem $1| \,|\sum_j w_j f(C_j)$ (with f non-decreasing).

2 Model, definitions, and preliminaries

2.1 Problem definition

We consider two types of scheduling problems. In both cases we are given a set of jobs $J = \{1,\ldots,n\}$ with work volume (processing time at speed 1) $v_j \geq 0$ and weights $w_j \geq 1$. We seek a schedule on a single machine (permutation of jobs) that minimizes the sum of weighted completion times. The speed of the machine may vary—this is where the problems distinguish.

In the problem *scheduling on a machine of given varying speed* we assume that the speed function $s : \mathbb{R}_+ \to \mathbb{R}_+$ is given indirectly by an oracle. Given a value v, the oracle returns the first point in time when the machine can finish v units of work. That is, for a speed function s the oracle returns the value

$$f(v) := \inf \left\{ b > 0 : \int_0^b s(t) \geq v \right\}.$$

Thus, for a given order of jobs or for given start times, we can compute the execution time of each job and then the total cost of the solution (assuming that there is no idle time).

In the problem *scheduling with speed-scaling* an algorithm determines not only a schedule for the jobs but will also decide at which speed $s \geq 0$ the machine will run at any time. Running a machine at certain speed requires a certain amount of power. Power is typically modeled as a monomial (convex) function of speed, $P(s) = s^\alpha$, with a small constant $\alpha > 1$. In applications, α is usually 2 or 3. Given an energy budget E, we ask for the optimal power (and thus speed)

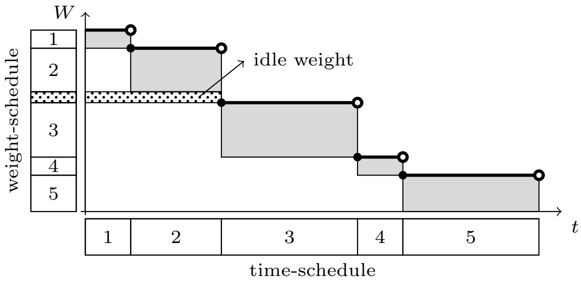

Figure 1: **2D-Gantt chart**. The x-axis shows a schedule, while the y-axis corresponds to $W(t) = \sum_{C_j > t} w_j$ plus the idle weight in the corresponding weight-schedule.

distribution and corresponding schedule such that the weighted sum of completion times of all jobs is minimized. More generally, we are interested in quantifying the tradeoff between the scheduling objective $\sum_{j \in J} w_j C_j$ and the total energy consumption, that is, we aim for computing the Pareto curve for the bicriteria minimization problem. We consider two variants of speed-scaling: If the machine can run at an arbitrary speed level $s \in \mathbb{R}_+$, we say that we are in the *continuous-speed* setting. On the other hand, if that machine can only choose among a finite set of speeds $\{s_1, \ldots, s_\kappa\}$, we are in an *discrete-speed* environment.

2.2 From time-space to weight-space

For a schedule $\mathcal{S}$, we let $C_j(\mathcal{S})$ denote the completion time of j and we let $W^{\mathcal{S}}(t)$ denote the total weight of jobs completed (strictly) after t. Note that by definition $W^{\mathcal{S}}(t)$ is right-continuous, i. e., if $C_j(\mathcal{S}) = t$, the weight of j does not count towards the remaining weight $W^{\mathcal{S}}(t)$. Whenever $\mathcal{S}$ is clear from the context we omit it. It is not hard to see that

$$\sum_{j \in J} w_j C_j(\mathcal{S}) = \int_0^\infty W^{\mathcal{S}}(t) dt. \tag{1}$$

Our main idea is to describe our schedule in terms of the remaining weight function W. That is, instead of determining C_j for each job j, we will implicitly describe the completion time of a job j by the value of W at the time that j completes. We call this value the *starting weight* of the job j, and denote it by S_j^w. Similarly, we define the *completion weight* of j as $C_j^w := S_j^w + w_j$. This has a natural interpretation in the 2D-Gantt chart (see Figure 1): A typical schedule determines completion times for jobs in *time-space* (x-axis), which is highly sensitive when the speed of the machine may vary. We call such a solution a *time-schedule*. Describing a scheduling solution in terms of remaining weight can be seen as scheduling in the *weight-space* (y-axis), yielding a *weight-schedule*. In weight-space the weights play the role of processing times. All notions that are usually considered in schedules apply in weight-space. For example, we say that a weight-schedule is feasible if there are no two jobs overlapping, and that the machine is idle at weight value w if $w \notin [S_j^w, C_j^w]$ for all j. In this case we say that w is *idle weight*.

A weight-schedule immediately defines a non-preemptive time-schedule by ordering the jobs on decreasing completion weights. Consider a weight-schedule $\mathcal{S}$ with completion weights $C_1^w \geq \ldots \geq C_n^w$, and corresponding completion times $C_1 \leq \ldots \leq C_n$. To simplify notation let $C_0 = C_{n+1}^w = 0$. Then we define the cost of $\mathcal{S}$ as $\sum_{j=1}^n (C_{j+1}^w - C_j^w) C_j$. It is easy to check, even from the 2D-Gantt chart, that this value equals $\sum_{j=1}^n x_j^{\mathcal{S}} C_j^w$, where $x_j^{\mathcal{S}}$ is the execution time of job j (in time-space). Moreover, the last expression is equivalent to Equation (1) if and only if the weight-schedule does not have any idle weight. In general, the cost of the weight-schedule can only overestimate the cost of the corresponding schedule in time space, given by (1).

On a machine of varying speed, the weight-schedule has a number of technical advantages. For instance, while creating idle *time* can increase the cost arbitrarily, we can create idle *weight*

without provoking an unbounded increase in the cost. This idea yields a simple rule to delay one or more jobs in the time-schedule without increasing the cost. More precisely, we have the following crucial observation that can be easily seen in the 2D-Gantt chart.

Observation 1. *Consider a weight-schedule $\mathcal{S}$ with enough idle weight so that decreasing the completion weight of some job j yields a feasible weight-schedule. This changes the time-schedule, but the completion* time *of each job $j' \neq j$ is not increased. Thus, this operation does not increase the total cost of $\mathcal{S}$.*

We remark that this idea might not work if the completion weight of j is increased instead of decreased.

3 A PTAS for scheduling on a machine with given speeds

In what follows we give a PTAS for minimizing $\sum_j w_j C_j$ on a machine with a given speed function. In order to gain structure, we start by applying several modifications to the instance and optimal solution. First we round the weights of the jobs to the next integer power of $1+\varepsilon$. This can only increase the objective function by a $1+\varepsilon$ factor.

Additionally, we discretize the weight-space in intervals that increase exponentially. That is, we consider intervals $I_u = [(1+\varepsilon)^{u-1}, (1+\varepsilon)^u)$ for $u \in \{1,\ldots,\nu\}$ where $\nu := \left\lceil \log_{1+\varepsilon} \sum_{j\in J} w_j \right\rceil$. We denote the length of each interval I_u as $|I_u| := \varepsilon(1+\varepsilon)^{u-1}$. We will apply two important procedures to modify a schedule in weight-space. These techniques are useful in order to create idle weight by only increasing a $1+\mathcal{O}(\varepsilon)$ factor the cost. This allows us to apply Observation 1. Similar techniques, applied in time-space, were used by Afrati et al. [1] for problems on constant-speed machines.

1. *Weight Stretch:* We multiply by $1+\varepsilon$ the completion weight of each job. This creates an idle weight interval of length εw_j before the starting weight of job j. This operation increases the cost by a $1+\varepsilon$ factor.
2. *Stretch Intervals:* The goal of this procedure is to create idle weight in each interval I_u. To do so we delay the completion weight of each job j with $C_j^w \in I_u$ by $|I_u|$, so that C_j^w belongs to I_{u+1}. Moreover, the amount of weight that job j was processing in I_u equals the amount of weight that is processed in I_{u+1} after modifying the schedule. This means that $|I_{u+1}| - |I_u| = \varepsilon^2(1+\varepsilon)^{u-1} = \varepsilon|I_{u+1}|/(1+\varepsilon)$ units of weight are left idle in I_{u+1} after the transformation, unless there was only one job completely covering I_{u+1}. By moving jobs within I_u, we can assume that this idle weight is consecutive. This transformation increases the cost by at most a factor $(1+\varepsilon)^2 = 1+\mathcal{O}(\varepsilon)$. Moreover, if some interval I_u contains the starting weight of some job, then I_u has now at least $|I_u| - |I_{u-1}|$ idle weight.

3.1 Dynamic program

Before giving further structure to near-optimal solutions, we explain our approach for obtaining a PTAS: we first describe a dynamic programming (DP) table with exponentially many entries and then discuss how to reduce its size.

Consider a subset of jobs $S \subseteq J$ and a partial schedule of S in the weight-space. In our dynamic program, S will correspond to the set of jobs at the beginning of the weight-schedule, i. e., if $j \in S$ and $k \in S \setminus J$ then $C_j^w < C_k^w$. A partial weight-schedule $\mathcal{S}$ of jobs in S implies a schedule in time-space with the following interpretation. Note that the makespan of the time-schedule is completely defined by the total work volume $\sum_j v_j$. We impose that the last job of the schedule, which corresponds to the first job in $\mathcal{S}$, finishes at the makespan. This uniquely determines a value of C_j for each $j \in S$, and thus also its execution time $x_j^{\mathcal{S}}$. The total cost of this partial schedule is $\sum_{j\in S} x_j^{\mathcal{S}} C_j^w$ (which has a simple interpretation in the 2D-Gantt chart).

Consider $\mathcal{F}_u := \{S \in J : w(S) \le (1+\varepsilon)^u\}$. That is, a set $S \in \mathcal{F}_u$ is a potential set to be schedule in I_u or before. For a given interval I_u and set $S \in \mathcal{F}_u$, we construct a table entry

$T(u, S)$ with a $(1 + \mathcal{O}(\varepsilon))$-approximation to the optimal cost of a weight-schedule of S subject to $C_j^w \leq (1 + \varepsilon)^u$ for all $j \in S$.

Consider now $S \in \mathcal{F}_u$ and $S' \in \mathcal{F}_{u-1}$ with $S' \subseteq S$. Let $\mathcal{S}$ be a partial schedule of S where the set of jobs with completion weight in I_u is exactly $S \setminus S'$. We define $\mathrm{APX}_u(S', S) = (1 + \varepsilon)^u \sum_{j \in S \setminus S'} x_j^{\mathcal{S}}$, which is a $(1 + \varepsilon)$-approximation to $\sum_{j \in S \setminus S'} x_j^{\mathcal{S}} C_j^w$, the partial cost associated to $S \setminus S'$. We remark that the values $\sum_{j \in S \setminus S'} x_j^{\mathcal{S}}$ and $\mathrm{APX}_u(S', S)$ do not depend on the whole schedule $\mathcal{S}$, but only on the total work volume of jobs in S'.

We can compute $T(u, S)$ with the following formula,

$$T(u, S) = \min\{T(u - 1, S') + \mathrm{APX}_u(S', S) : S' \in \mathcal{F}_{u-1}, S' \subseteq S\}.$$

This DP table contains an exponential number of entries, since $\mathcal{F}_u$ can be of exponential size. In the following we show that we can compute a set $\tilde{\mathcal{F}}_u$ of polynomial size that yields a $(1 + \varepsilon)$-approximation to the solution. We remark that the set $\tilde{\mathcal{F}}_u$ will be *universal*, that is, the set is completely independent of the speed of the machine. Therefore, the same set can be used in the speed-scaling scenario.

3.2 Light jobs

We structure an instance by classifying jobs by their size in weight-space.

Definition 2. *In a given schedule, a job j is said to be* light *if $w_j \leq \varepsilon |I_u|$, where u is such that $S_j^w \in I_u$. A job that is not light is* heavy.

Given a weight-schedule for heavy jobs, we can greedily find a $(1+\mathcal{O}(\varepsilon))$-approximate solution for the complete instance. To show this, consider any weight-schedule $\mathcal{S}$. First, remove all light jobs. Then we move jobs within each interval I_u, such that the idle weight inside each interval is consecutive. Clearly, this can only increase the cost of the solution by a $1 + \varepsilon$ factor. After, we apply the following preemptive greedy algorithm to assign light jobs.

Algorithm Smith in Weight-Space

1. For $u = 1, \ldots, \nu$ and each idle weight $w \in I_u$, process a job j maximizing v_j/w_j among all available jobs with $w_j \leq \varepsilon |I_u|$.

To remove preemptions, we apply the Stretch Interval subroutine[1] twice, creating an idle weight interval in I_u of length at least $2\varepsilon |I_u|/(1 + \varepsilon) \geq \varepsilon |I_u|$ (for $\varepsilon \leq 1$). This gives enough space in each interval I_u to completely process the (unique) preempted light job with starting weight in I_u. Then, Observation 1 implies that we can remove preemptions, obtaining a new schedule $\mathcal{S}'$. We now show that the cost of $\mathcal{S}'$ is at most a factor of $1 + \mathcal{O}(\varepsilon)$ larger than the cost of $\mathcal{S}$. To do so we need a few definitions.

For any weight-schedule $\mathcal{S}$, let us define the *remaining volume function* as

$$V^{\mathcal{S}}(w) := \sum_{j : C_j^w \geq w} v_j.$$

For a given w, let $I_j(w)$ be equal 1 if the weight-schedule processes j at weight w, and 0 otherwise. Then, $f_j(w) := (1/w_j) \int_w^\infty I_j(w') dw'$ corresponds to the fraction of job j processed after w. With this we define the *fractional remaining volume function*, which is similar to the remaining volume function but treats light jobs as "liquid":

$$V_f^{\mathcal{S}}(w) := \sum_{j : j \text{ is light}} f_j(w) \cdot v_j + \sum_{j : j \text{ is heavy}, C_j^w \geq w} v_j \qquad \text{for all } w \geq 0.$$

We notice that $V_f^{\mathcal{S}}(w) \leq V^{\mathcal{S}}(w)$ for all $w \geq 0$.

[1]The Stretch Interval procedure also applies to preemptive settings by interpreting each piece of a job as an independent job.

Consider now the function $f(v)$ corresponding to the earliest point in time in which the machine can process a work volume of v, i. e.,

$$f(v) := \inf \left\{ b \geq 0 : \int_0^b s(t)dt \geq v \right\} \qquad \text{for all } v \geq 0.$$

Notice that this is the same function used when transforming our problem to $1| \,|\sum_j w_j f(C_j)$. It is easy to see—even from the 2D-Gantt chart—that $\int_0^\infty f(V^{\mathcal{S}}(w))dw$ corresponds to the cost of the weight-schedule $\mathcal{S}$. Also, notice that $f(v)$ is non-decreasing, so that $V^{\mathcal{S}}(w) \leq V^{\mathcal{S}'}(w)$ for all w implies that the cost of $\mathcal{S}$ is at most the cost of $\mathcal{S}'$.

Lemma 3. *The cost of $\mathcal{S}'$ is at most $1 + \mathcal{O}(\varepsilon)$ times larger than the cost of $\mathcal{S}$.*

Proof. Consider the weight-schedule $\mathcal{S}_H$ of heavy jobs before running Algorithm Smith in Weight-Space. Let $\mathcal{S}_f$ be the preemptive schedule obtained after applying such algorithm. First we observe that for any given $w \geq 0$, $V_f^{\mathcal{S}_f}(w)$ is a lower-bound on $V_f^{\mathcal{S}''}(w)$ for any schedule $\mathcal{S}''$ that coincides with $\mathcal{S}_H$ on the heavy jobs. This follows by a simple exchange argument, since the greedy Smith-type rule in algorithm chooses the job that packs as much possible volume in the available weight among all light jobs.

Observe that applying Stretch Interval twice can increase the starting weight of a job by at most a factor $(1+\varepsilon)^4$. Also, applying this procedure to $\mathcal{S}_f$ only moves pieces of jobs forwards. This implies that $V_f^{\mathcal{S}'}(w) \leq V_f^{\mathcal{S}_f}((1+\varepsilon)^{-4}w)$ for all w. Moreover, each light job in $\mathcal{S}'$ completely finishes within an interval I_u. Thus, $V^{\mathcal{S}'}(w) \leq V_f^{\mathcal{S}_f}((1+\varepsilon)^{-1}w)$. We conclude that

$$V^{\mathcal{S}'}((1+\varepsilon)^5 w) \leq V_f^{\mathcal{S}'}((1+\varepsilon)^4 w) \leq V_f^{\mathcal{S}_f}(w) \leq V_f^{\mathcal{S}}(w) \leq V^{\mathcal{S}}(w) \qquad \text{for all } w \geq 0.$$

Taking the function $f(\cdot)$ on this sequence of inequalities and integrating implies that

$$\int_0^\infty f(V^{\mathcal{S}'}((1+\varepsilon)^5 w))dw \leq \int_0^\infty f(V^{\mathcal{S}}(w))dw.$$

Finally, the right hand side of this inequality is the cost of $\mathcal{S}$, and a simple change of variables implies that the left hand side is $(1+\varepsilon)^{-5}$ times the cost of $\mathcal{S}'$. The lemma follows. ☐

The next result follows directly from our previous result.

Corollary 4. *At a loss of a $1+\mathcal{O}(\varepsilon)$ factor in the objective function, we can assume the following. For a given interval I_u, consider any pair of jobs j, k whose weights are at most $\varepsilon|I_u|$. If both jobs are processed in I_u or later and $v_k/w_k \leq v_j/w_j$, then $C_j^w \leq C_k^w$.*

3.3 Localization

The objective of this section is to compute, for each job $j \in J$, two values r_j^w and d_j^w so that job j is scheduled completely within $[r_j^w, d_j^w)$ in some $(1+\mathcal{O}(\varepsilon))$-approximate weight-schedule. We call r_j^w and d_j^w the *release-weight* and *deadline-weight* of job j, respectively. Crucially, we need that the length of the interval $[r_j^w, d_j^w)$ is not too large, namely that $d_j \in \mathcal{O}(\text{poly}(1/\varepsilon) r_j)$. Such values can be obtained by using Corollary 4 and techniques from [1]. The release- and deadline-weights will help us finding a compact set $\tilde{\mathcal{F}}_u$.

We consider an initial value for r_j^w and then increase its value iteratively. We will restrict ourselves to values of r_j^w that are integer powers of $1+\varepsilon$. Consider an arbitrary weight-schedule. Applying the Weight Stretch subroutine, we obtain that the starting weight of each job satisfies $S_j^w \geq \varepsilon w_j$. Thus, we can safely define r_j^w as εw_j rounded down to an integer power of $1+\varepsilon$. We can now use the approach taken in [1]. Let J_u be the set of all jobs with r_j^w equal to $(1+\varepsilon)^{u-1}$. We partition J_u into light and heavy jobs, and then classify heavy jobs by its weight. Note that a heavy job in J_u can have weights w with $\varepsilon|I_u| < w \leq 1/\varepsilon(1+\varepsilon)^{u-1}$, where the last inequality follows since $r_j^w \geq \varepsilon w_j$. Therefore for a fix u we only need to consider heavy jobs with a weight $w \in \Omega_u := \{(1+\varepsilon)^i : \varepsilon|I_u| < (1+\varepsilon)^i \leq 1/\varepsilon(1+\varepsilon)^{u-1}\}$. Crucially, note that $|\Omega_u| \in \mathcal{O}(\log_{1+\varepsilon} 1/\varepsilon) \subseteq \mathcal{O}(1/\varepsilon \cdot \ log 1/\ e)$. Based on this we give the following decomposition of the set of jobs with a given released weight.

Definition 5. *Given release weights for each job, we define* $J_u = \{j : r_j^w = (1+\varepsilon)^{u-1}\}$. *Additionally, we decompose* J_u *into a set of light jobs* $L_u := \{j \in J_u : w_j \leq \varepsilon|I_u|\}$, *and a set* $H_{u,w} = \{j \in J_u : w_j = w\}$ *of heavy jobs of weight* w *for each* $w \in \Omega_u$.

Let us now consider all jobs in L_u. If $w(L_u)$ is more than $|I_u|$ then some of these jobs will have to start in I_{u+1}. By Corollary 4 we can choose the set of possible jobs with starting weight in I_u greedily, and increase the release weight of the rest. Similarly, since the weight of each job in $H_{u,w}$ is the same, we can always give priority to jobs with the largest processing time. With this idea we have the following lemma.

Lemma 6. *We can compute in polynomial time release-weights* r_j^w *for each job* j *such that there exists a* $(1+\mathcal{O}(\varepsilon))$*-approximate weight-schedule satisfying the release weights and for all interval* I_u, $w(J_u) \in \mathcal{O}(1/\varepsilon^3 \cdot \log 1/\varepsilon \cdot |I_u|)$.

Proof. Initialize r_j^w as εw_j rounded down to an integer power of $(1+\varepsilon)$. Let J_u be the set of jobs with release weights $(1+\varepsilon)^{u-1}$. Decompose set J_u in L_u and sets $H_{u,w}$ for all $w \in \Omega_u$ as above. We consider jobs in L_u and each set $H_{u,w}$ separately.

By Corollary 4 we know that within an interval I_u we can order light jobs and processed first the job with largest v_j/w_j ratio. Thus, if the total weight of jobs in L_u is larger than $(1+\varepsilon)|I_u|$ we can simply increase the release weight of the jobs with the largest w_j/v_j to $(1+\varepsilon)^u$. We do this until $w(L_u) \leq (1+\varepsilon)|I_u|$.

We do a similar technique for jobs in $H_{u,w}$. If $w(H_{u,w}) > |I_u| + w$ and $|H_{u,w}|$ contains more than one job, then we can delay the release weight of the jobs with smallest v_j. This follows by a simple interchange argument, since if there are two jobs with the same weight then the one with smallest work has smaller (larger) completion time (weight) .

With this we obtain a set $H_{u,w}$ with

$$w(H_{u,w}) \leq |I_u| + w \leq |I_u| + \frac{1}{\varepsilon}(1+\varepsilon)^{u-1} \in \mathcal{O}(1/\varepsilon^2) \cdot |I_u|.$$

We can iterate the two procedures described above until the following property holds: for all u and $w \in \Omega_u$ we have that $w(L_u) \leq (1+\varepsilon)|I_u|$ and $w(H_{u,w}) \in \mathcal{O}(1/\varepsilon^2) \cdot |I_u|$. The result follows since $|\Omega_u| \in \mathcal{O}(1/\varepsilon \cdot \log 1/\varepsilon)$. ☐

We use the previous lemma to define the deadline-weights by using the following idea. For s large enough (but constant), Stretch Intervals creates enough idle weight in I_{u+s} to fit all jobs release at $(1+\varepsilon)^u$ that have not yet finished by $(1+\varepsilon)^{u+s+1}$. This allows us to apply Observation 1.

Lemma 7. *We can compute in poly-time values* r_j^w *and* d_j^w *for each* $j \in J$ *such that: (i) there exists a* $(1+\mathcal{O}(\varepsilon))$*-approximate weight-schedule that processes each job* j *within* $[r_j^w, d_j^w)$, *(ii) there exists a constant* $s \in \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ *such that* $d_j^w \leq r_j^w \cdot (1+\varepsilon)^s$, *(iii)* r_j^w *and* d_j^w *are integer powers of* $(1+\varepsilon)$, *and (iv) the values* r_j^w *an* d_j^w *are independent of the speed of the machine.*

Proof. Consider the release weights given by the previous lemma and consider the associated sets J_u for each u. The construction of the release weights guarantees the existence of a value $s \in \mathcal{O}(\log_{1+\varepsilon}(1/\varepsilon^4 \cdot \log 1/\varepsilon)) \subseteq \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ such that $w(J_u) \leq \varepsilon|I_{u+s-1}|/(1+\varepsilon)$.

Now we apply Stretch Intervals to the schedule. This creates $\varepsilon|I_{u+s-1}|/(1+\varepsilon)$ idle weight in interval I_{u+s-1}, unless there was one job completely covering I_{u+s-1}. If that is not the case, then we can move all jobs in J_u with starting weight in I_{u+s} or larger to be completely processed inside I_{u+s-1}. By Observation 1, doing this can only increase the objective function by a $1+\mathcal{O}(\varepsilon)$ factor. Similarly, if there was a job k completely covering I_{u+s-1}, then the idle weight that I_{u+s-1} should have contained can be considered to be just before the starting weight of k. In this case we can move all jobs in J_u that were being processed after I_{u+s-1} to just before S_k^w.

In either case we constructed a solution where each job in J_u is completely processed in $[(1+\varepsilon)^{u-1}, (1+\varepsilon)^{u+s-1})$. We conclude by defining $d_j^w = (1+\varepsilon)^{u+s}$ for each job $j \in J_u$. Note that (iv) also follows since r_j^w and d_j^w defined as before do not depend on the machine's speed. ☐

3.4 Compact Search Space

Recall the definition of $\mathcal{F}_u$ given at the beginning of Section 3.1, and that we need to define a polynomial size version of $\mathcal{F}_u$. We call this set $\tilde{\mathcal{F}}_u$. Instead of describing a set $S \in \tilde{\mathcal{F}}_u$, we describe $V = J \setminus S$, that is, the jobs with completion weights in I_{u+1} or later. That is, we define a set $\mathcal{D}_u$ that will contain the complements of sets in $\tilde{\mathcal{F}}_u$. In order to define $\mathcal{D}_u$ we use the release- and deadline-weights given by Lemma 7. If $V \in \mathcal{D}_u$, then V must contain all jobs with release weight in I_{u+1} or after. Let $\overline{V} := \{j \in J : r_j^w \geq (1+\varepsilon)^u\}$.

Observation 8. *Each set $V \in \mathcal{D}_u$ is of the form $V' \cup \overline{V}$, where every job $j \in V'$ has $r_j^w \leq (1+\varepsilon)^{u-1}$.*

Thus we only need to describe all possibilities for V'. For a job $j \in V'$ we can assume that $d_j^w \geq (1+\varepsilon)^{u+1}$. Therefore, by Lemma 7, we have that $r_j^w \geq (1+\varepsilon)^{u+1-s}$, where $s \in \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$.

Observation 9. *Each set $V \in \mathcal{D}_u$ is of the form $\left(\bigcup_{v=u+1-s}^{u-1} V_v'\right) \cup \overline{V}$, where $V_v' := \{j \in V' : r_j^w = (1+\varepsilon)^v\}$.*

Then, we aim to find a collection of subsets that can play the role of V_v'. If the size of this collection is at most a polynomial number k, we could conclude that $|\mathcal{D}_u| \leq k^s = k^{\mathcal{O}(\log(1/\varepsilon)/\varepsilon)}$.

In order to do so, recall that J_v denote all jobs with release-weights equal to $(1+\varepsilon)^v$, and that we can write $J_v = L_v \cup (\bigcup_w H_{v,w})$ where w ranges within $\mathcal{O}(\log_{1+\varepsilon} 1/\varepsilon)$ many values. Thus, defining $V_{v,w}' := V_v' \cap H_{v,w}$ we can further decompose V_v' as $(V_v' \cap L_v) \cup (\bigcup_w V_{v,w}')$. Now notice that $V_{v,w}'$ is a subset of $H_{v,w}$ which, as justified in the next observation, contains constantly many sets.

Observation 10. *Without loss of generality, we can restrict ourselves to consider sets $V_{v,w}'$ among $\mathcal{O}(1/\varepsilon)$ distinct options.*

Proof. Each job in $H_{v,w}$ for $w \in \Omega_v$, and the total weight of $H_{v,w}$ is at most $|I_v| + w$, thus $H_{v,w}$ contains at most $1 + |I_v|/w$ many jobs. Since by definition of $H_{v,w}$ we have that $w \geq \varepsilon|I_v|$, we obtain that $|H_{v,w}| \in \mathcal{O}(1/\varepsilon)$. Moreover, all jobs in $H_{v,w}$ has the same weight w and the same release-weight. Therefore, we know that this jobs are order by their work volume in an optimal solution. Thus, we can restrict ourselves to sets $V_{v,w}'$ that respect this order. The observation follows since there are at most $|H_{v,w}| \in \mathcal{O}(1/\varepsilon)$ many sets that respect this order. $\square$

Given v, the index w ranges over $|\Omega_v| \in \mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ many values. Thus the following holds.

Observation 11. *For each v the set $\bigcup_w V_{v,w}'$ can be chosen over $(1/\varepsilon)^{\mathcal{O}(\log(1/\varepsilon)/\varepsilon)} = 2^{\mathcal{O}(\log(1/\varepsilon)^2/\varepsilon)}$ many alternatives.*

We use a similar argument for $V_v' \cap L_v$. Indeed, $w(L_v) \leq (1+\varepsilon)|I_v|$ and jobs in L_v will be processed as light jobs (by Lemma 7). We now show that we can group light jobs together in order to diminish the possibilities for L_v. This is done as follows. Let s be the number given by Lemma 7. We order jobs in L_v by inverse Smith's rule order. Then we greedily find groups of jobs in L_v by going in the list of jobs from left to right such that each group has total weight in $[\varepsilon|I_v|/s, 2\varepsilon|I_v|/s]$. Recalling that $w(L_v) \in \mathcal{O}(|I_v|)$, we obtain that this procedures creates at most $\mathcal{O}(s/\varepsilon)$ groups. Let $L_{v,i}$ be the i-th groups.

Lemma 12. *There exists a $(1+\mathcal{O}(\varepsilon))$-approximate weight schedule such that: (i) it satisfies the release- and deadline-weights of Lemma 7, (ii) in each group $L_{v,i}$ all jobs are processed consecutively for each v, i, and (iii) within each set L_v jobs are processed following reverse Smith's rule order.*

Proof. Consider the schedule given by Lemma 7. First notice that this schedule must not necessarily satisfy the property of Corollary 4, since in the proof of Lemma 7 we changed the order of jobs. However, in that proof for each v we moved jobs in J_v together. Therefore, within each J_v the reverse Smith's rule order is preserved. Let us fix an interval $I_{v'}$. Within this interval, the schedule only processes jobs in J_v with $v \in \{v' - s, \ldots, v'\}$. Let us fix such a set J_v. Since we follow reverse Smith's rule within J_v, there is at most two sets $L_{v,i}$ that are partially processed in $I_{v'}$. They require $4\varepsilon|I_v|/s$ extra space within $I_{v'}$ in order to be processed completely within $I_{v'}$. Summing over $v \in \{v' - s, \ldots, v' - 1\}$, we obtain that in total we require

$$4\varepsilon \sum_{v=v'-s}^{v'-1} \frac{|I_v|}{s} \leq 4\varepsilon s \frac{|I_{v'}|}{s} \in \mathcal{O}(\varepsilon |I_{v'}|)$$

extra space in $I_{v'}$. The result follows since we can create enough idle time within $I_{v'}$ by applying $\mathcal{O}(1)$ times the procedure Stretch Intervals. We remark that the procedure described works *simultaneously* for all intervals I_v. □

With this lemma, we can find a compact description to $V'_v \cap L_v$. Indeed, to specify $V'_v \cap L_v$, i. e., the jobs in L_v that are processed in I_{u+1} or later, we just need to determine the index i such that jobs in $L_{v,j}$ with $j \geq i$ are V'_v and jobs in $L_{v,j}$ with $j < i$ are not in V'_v. Since i ranges over $\mathcal{O}(s/\varepsilon) \in \mathcal{O}(\log(1/\varepsilon)/\varepsilon^2)$ many options, we obtain the following.

Observation 13. *The set $V'_v \cap L_v$ can be chosen over $O(\log(1/\varepsilon)/\varepsilon^2)$ different options.*

Combining this last observations and Observation 11, we obtain that V'_v can take at most $k \leq 2^{\mathcal{O}(\log^2(1/\varepsilon)/\varepsilon)}$ many different options. By Observation 9, we conclude that V' belongs to a set of size at most $k^s \leq 2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)}$. With this and Observation 8, we can define $\mathcal{D}_u$ having size at most $2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)}$. With our discussion we conclude the following lemma.

Lemma 14. *For each interval I_u, we can construct in poly-time a set $\tilde{\mathcal{F}}_u$ that satisfies the following: (i) there exists a $(1 + \mathcal{O}(\varepsilon))$-approximate weight-schedule in which the set of jobs with completion weight at most $(1+\varepsilon)^u$ belongs to $\tilde{\mathcal{F}}_u$ for each interval I_u, (ii) the set $\tilde{\mathcal{F}}_u$ has cardinality at most $2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)}$, and (iii) the set $\tilde{\mathcal{F}}_u$ is completely independent of the speed of the machine.*

With this lemma and the discussion at the beginning of this section we obtain a PTAS, which is best possible from an approximation point of view, since the problem is known to be strongly NP-hard [12].

Theorem 15. *There exists an efficient PTAS for minimizing the weighted sum of completion times on a machine with given varying speed.*

Proof. We just need to argue about the running time. It is easy to see that the time for creating sets $\tilde{\mathcal{F}}_u$ is dominated by the time needed to solve the dynamic program. Moreover, the number of entries of the table is $2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)} \cdot \log(\sum_j w_j)$, and the time needed to fill each entry is $2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)} \cdot n$. Therefore the running time is $2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)} \cdot \log(\sum_j w_j) \cdot 2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)} \cdot n = 2^{\mathcal{O}(\log^3(1/\varepsilon)/\varepsilon^2)} \cdot \log(\sum_j w_j) \cdot n$ □

4 Speed-scaling for continuous speeds

We now consider the problem of minimizing $\sum_j w_j C_j$ for a given amount of energy available E. In this section we consider a continuous spectrum of speeds. It is easy to see that in this setting each job will be executed at a uniform speed because of the convexity of the power function [23]. Let s_j be the speed at which job j is running. Then j's power consumption is $p_j = s_j^\alpha$, and its execution time is $x_j = v_j/s_j = v_j/p_j^{1/\alpha}$. The energy that is required for processing j is $E_j = p_j \cdot x_j = p_j \cdot \frac{v_j}{s_j} = s_j^{\alpha-1} \cdot v_j = v_j^\alpha / x_j^{\alpha-1}$.

The problem of computing the optimal energy (or speed) assignment for all jobs in a given job sequence π using a total amount of energy E can be expressed as a convex program: Let jobs be indexed according to some given order π. We rewrite the objective function as $\sum_{j=1}^n w_j C_j = \sum_{j=1}^n w_j \sum_{k=1}^j x_k = \sum_{j=1}^n x_j \sum_{k=j}^n w_k$ and define $W_j = \sum_{k=j}^n w_k$. Note that $x_j = \left(v_j^\alpha / E_j\right)^{1/(\alpha-1)}$. Then the problem can be formulated as

$$\min \quad \sum_{j=1}^n W_j \cdot \left(\frac{v_j^\alpha}{E_j}\right)^{1/(\alpha-1)} \tag{2}$$

$$\sum_{j=1}^n E_j \leq E \tag{3}$$

$$E_j \geq 0, \qquad\qquad j \in \{1, \ldots, n\} \tag{4}$$

This program has linear constraints and a convex objective function (which follows from the convexity of the function $1/x$). Such programs can be solved in polynomial time up to an arbitrary precision [18] with the Ellipsoid method. However, with a better understanding of its structure and by applying the well-known KKT conditions, we easily derive an explicit formulae for computing an optimal energy assignment.

The problem (2)–(4) has a feasible solution since $E_j = 0$, for $j = 1, \ldots, n$ satisfies both constraints, (3) and (4). However, an optimal solution satisfies Constraint (3) with equality, since we allow arbitrary speeds and thus arbitrary energy assignments, and the smallest increase in the assigned energy will decrease the total cost. For the same reason and with a positive energy budget, an optimal solution will never assign zero energy to any job; hence, none of the Inequalities (4) is satisfied with equality.

Now, we apply the well-known Karush-Kuhn-Tucker (KKT) conditions which give necessary and sufficient conditions on an optimal solution to convex programs with linear constraints [6]. With the observations above they reduce to the following conditions.

Lemma 16 (KKT conditions)**.** *A vector $(E_1, \ldots, E_n)$ is an optimal solution to the convex program* (2)–(4) *if and only if*

(a) $(E_1, \ldots, E_n)$ is feasible and satisfies (3) *with equality but none of* (4) *with equality, and*

(b) there exists a parameter $\lambda \geq 0$ such that $\nabla g(E_1, \ldots, E_n) + \lambda \cdot \mathbb{1} = 0$.

Here, $\mathbb{1}$ denotes a vector with ones in each coordinate.

These conditions lead to an explicit description of the optimal solution of the convex program (2)–(4).

Theorem 17. *For a given job sequence π, a power function $P(s) = s^\alpha$ and an energy budget E, the optimal energy assignment in an optimal schedule for minimizing $\sum_j w_j C_j$ subject to $C_{\pi(1)} < \ldots < C_{\pi(n)}$ is determined by*

$$E_j = v_j \cdot W_j^{(\alpha-1)/\alpha} \cdot \frac{E}{\gamma_\pi}, \qquad \textit{where } \gamma_\pi = \sum_{j=1}^n v_{\pi(j)} \cdot W_{\pi(j)}^{(\alpha-1)/\alpha}.$$

Proof. To simplify notation let us reorder the jobs so that $C_1 < C_2 < \ldots < C_n$ and $\pi(j) = j$ for all j (as above). Then we need to show that the optimal energy assignment is given by

$$E_j = v_j \cdot W_j^{(\alpha-1)/\alpha} \cdot \frac{E}{\gamma}, \qquad \text{where } \gamma = \sum_{j=1}^n v_j \cdot W_j^{(\alpha-1)/\alpha}.$$

Let $(E_1, \ldots, E_n)$ be an optimal solution to the convex program (2)–(4). By Lemma 16(b), there is a $\lambda \geq 0$ such that for every job $j \in \{1, \ldots, n\}$ holds

$$W_j \cdot v_j^{\alpha/(\alpha-1)} \cdot \frac{-1}{\alpha - 1} \cdot E_j^{-\alpha/(\alpha-1)} + \lambda = 0\,,$$

which is equivalent to

$$E_j = v_j \cdot W_j^{(\alpha-1)/\alpha} \cdot \left(\frac{1}{(\alpha-1)\lambda}\right)^{(\alpha-1)/\alpha} . \tag{5}$$

To determine the lagrangean multiplier λ (or directly replace the last product term in (5)), we use the fact that Inequality (3) is satisfied with equality, that is,

$$E \;=\; \sum_{j=1}^{n} E_j \;=\; \sum_{j=1}^{n} v_j \cdot W_j^{(\alpha-1)/\alpha} \cdot \left(\frac{1}{(\alpha-1)\lambda}\right)^{(\alpha-1)/\alpha} \;=\; \gamma \cdot \left(\frac{1}{(\alpha-1)\lambda}\right)^{(\alpha-1)/\alpha} .$$

Then, we can express the values E_j in (5) independently of λ and conclude with

$$E_j = \frac{E}{\gamma} \cdot v_j \cdot W_j^{\frac{\alpha-1}{\alpha}} .$$

□

Interestingly, the optimal job sequence is independent of the energy distribution, and even stronger, it is independent of the overall energy budget. In other words, one scheduling sequence is universally optimal for all energy budgets. Furthermore, this sequence is obtained by solving *in weight-space* a (standard) scheduling problem with a cost function that depends on the power function.

Theorem 18. *Let $\alpha > 1$ be a constant. Given a power function $P(s) = s^\alpha$, there is a universal sequence that minimizes $\sum_j w_j C_j$ for any energy budget. The sequence is given by reversing an optimal solution of the scheduling problem $1| \,|\sum w_j C_j^{(\alpha-1)/\alpha}$.*

Proof. Assuming that jobs are indexed following a sequence π,

$$\sum_{j=1}^{n} w_j C_j(E) = \sum_{j=1}^{n} W_j \cdot \left(\frac{v_j^\alpha}{E_j}\right)^{\frac{1}{\alpha-1}} = \frac{1}{E^{\frac{1}{\alpha-1}}} \cdot \left(\sum_{j=1}^{n} v_j \cdot W_j^{\frac{\alpha-1}{\alpha}}\right)^{\frac{\alpha}{\alpha-1}} , \tag{6}$$

where the last equation comes from the definition of γ_π (see Thm. 17) and standard transformations. This equation implies that the optimal job sequence is independent of the available energy budget since E only plays a role in the factor outside the sum, which is independent of the permutation. Since the exponent is constant, the problem of finding the optimal sequence under an optimal energy-distribution reduces to finding the sequence that minimizes $\sum_{j=1}^{n} v_j \cdot W_j^{(\alpha-1)/\alpha}$. Now recall the reinterpretation that the 2D-Gantt chart view offers (see Sect. 2). Then this problem is equivalent to the scheduling problem in the *weight-space* with *varying speed on the weight-axis or general cost function in the weight-space.* This problem can be directly translated into minimizing the total weighted completion time on a machine with varying speed (or the desired form with a generalized cost function) by re-interpreting weight-space as time-space (Section 2). We simply define a new problem in time-space with processing times $v'_j = w_j$ and weight $w'_j = v_j$ with the objective function of minimizing $\sum_{j=1}^{n} w'_j \cdot f\left(\sum_{k=1}^{j} v'_j\right)$. This is a problem of the desired type. By Section 2 it is easy to see, that a solution π' to the new problem in time-space, has a corresponding solution π in the weight-space with same total cost; π is the reverse of π'. □

Thus, the scheduling part of the speed-scaling scheduling problem reduces to a problem which can be solved by our PTAS from Sect. 3. Also, since the cost function $f(x) = x^{(\alpha-1)/\alpha}$ is concave for $\alpha > 1$, the specialized PTAS in [21] also solves it. Combining Theorems 17 and 18 gives the main result.

Theorem 19. *There is a PTAS for the continuous speed-scaling and scheduling problem with a given energy budget E. Indexing jobs in the order of the solution given by the PTAS, the*

$(1+\varepsilon)$*-approximate Pareto curve describing the approximate scheduling cost as a function of the available energy is given by the right-hand-side of Equation* (6)*, i.e.,*

$$\operatorname{cost}(E) = \frac{1}{E^{\frac{1}{\alpha-1}}} \cdot \left(\sum_{j=1}^{n} v_j \cdot W_j^{\frac{\alpha-1}{\alpha}} \right)^{\frac{\alpha}{\alpha-1}} .$$

Proof. Define a new input instance with $v'_j = w_j$ and $w'_j = v_j$ and run a PTAS for the scheduling problem $1| \,|\sum w_j f(C_j)$ with $f : x \to x^{(\alpha-1)/\alpha}$ (Theorem 15 or [21]). Let π denote the reverse of the solution sequence. Theorem 18 guarantees that for a given optimal energy allocation, π has scheduling cost at most a factor $(1+\varepsilon)$ larger than the optimal sequence. This optimal energy allocation can be computed with the formulae in Theorem 17. The optimal scheduling cost as a function of the energy budget E are described by Equation (6). Since π is a $(1+\varepsilon)$-approximate solution this concludes the proof. □

5 Speed-scaling for discrete speeds

In this section we consider a more realistic setting, where the machine can choose one out of κ different speeds available $s_1 > \ldots > s_\kappa \geq 1$. For this problem we resolve the complexity status and show that it is NP-hard even when $\kappa = 2$. We give an FPTAS for this problem when the number of available speeds and their maximum ratio are bounded by a constant. For the general problem with arbitrarily many speed states we give PTAS even for arbitrary power functions.

5.1 A PTAS for discrete speeds

Let the power function $P(s)$ be an arbitrary computable function. To derive our algorithm, we adopt the PTAS for scheduling on a machine with given varying speed (Sect. 3) and incorporate the allocation of energy.

We adopt the same definitions of weight intervals I_u and sets $\mathcal{F}_u$ as in Sect. 3. We discretize the weight-space in intervals $I_u = [(1+\varepsilon)^{u-1}, (1+\varepsilon)^u)$ for $u \in \mathbb{Z}$ and denote the length of the interval as $|I_u| := \varepsilon(1+\varepsilon)^{u-1}$. For a subset of jobs $S \in \mathcal{F}_u$ and a value $z \geq 0$, let $E[u, S, z]$ be the minimum total energy necessary for scheduling S such that all completion weights are in interval I_u or before, i.e., $C_j^w \leq (1+\varepsilon)^u$ for each $j \in S$, and the scheduling cost is at most z, i.e., $\sum_{j \in S} x_j \cdot C_j^w \leq z$ where x_j is the execution time under some feasible speed assignment. Recall that the speed assignment determines the energy. The recursive definition of a state is as follows:

$$E(u, S, z) = \min\{E(u-1, S', z') + \mathrm{APX}_u(S \setminus S', z - z') : S' \in \mathcal{F}_{u-1}, S' \subseteq S\}.$$

Here $\mathrm{APX}_u(S \setminus S', z - z')$ is the minimum energy necessary for scheduling all jobs $j \in S \setminus S'$ with $C_j^w \in I_u$, such that their partial (rounded) cost $\sum_{j \in S \setminus S'} x_j (1+\varepsilon)^u$ is at most $z - z'$. This optimization problem can be solved by a linear program when aiming for a $(1+\varepsilon)$-approximation of the optimal scheduling cost. In that case, we can round all completion weights up to powers of $1+\varepsilon$. Thus, jobs in $S' \subseteq S$, $S' \in \mathcal{F}_{u-1}$ have the same completion weight $(1+\varepsilon)^u$. Then we can formulate the subproblem of computing $\mathrm{APX}_u(S \setminus S', z - z')$ as a linear program. Let the solution variable $\ell_i \geq 0$, $i \in \{1, \ldots, \kappa\}$, denote the length of the time interval in which the machine is running at speed s_i. These lengths must guarantee that the total processing volume $v(S' \setminus S)$

can be completed (Eq. (8)) and that the total scheduling cost does not exceed $z - z'$ (Eq. (9)).

$$\min \sum_{i=1}^{\kappa} \ell_i \cdot P(s_i) \tag{7}$$

$$\sum_{i=1}^{\kappa} \ell_i \cdot s_i = \sum_{j \in S \setminus S'} v_j \tag{8}$$

$$\sum_{i=1}^{\kappa} \ell_i \cdot (1+\varepsilon)^u \le z - z' \tag{9}$$

$$\ell_i \ge 0 \tag{10}$$

We let the DP fill the table for $u \in \{0, \ldots, \nu\}$ with $\nu = \lceil \log \sum_{j \in J} w_j \rceil$ and $z \in [1, z_{\mathrm{UB}}]$ for some upper bound such as $z_{\mathrm{UB}} = \sum_{j \in J} w_j \sum_{k=1}^{j} v_j / s_\kappa$. Then among all end states $[\nu, J, \cdot\,]$ with value at most the energy budget E we choose the one with minimum cost z. Then we obtain the corresponding $(1+\varepsilon)$-approximate solution for energy E by backtracking.

This DP has an exponential number of entries. However, we can apply results from Section 3 and standard rounding techniques to reduce the running time.

Theorem 20. *There is an efficient PTAS for minimizing the total scheduling cost for speed-scaling with a given energy budget.*

Proof. The DP computes a $(1+\varepsilon)$-approximation in exponential time. In Lemma 14, we showed how to reduce the exponential number of subsets in $\mathcal{F}_u$ to a polynomial number at the cost of a factor $1 + \mathcal{O}(\varepsilon)$ in the total scheduling cost. Recall that the sets $\tilde{\mathcal{F}}_u$ given by that lemma are independent of the speed of the machine. Therefore we can use these sets directly in our setting.

It remains to reduce the number of possible values of cost $z \in [0, z_{\mathrm{UB}}]$. At the cost of a factor $1 + \varepsilon$, we may round up in each state the scheduling cost to next integer power of $1 + \delta$ with $\delta = (1+\varepsilon)^{1/\nu} - 1$. In each state transition of the DP, we loose up to a factor $1 + \delta$ in the scheduling cost, which amounts to at most a factor $(1+\delta)^\nu = 1 + \varepsilon$ under ν state transitions. When restricting to powers of $1+\delta$ then the number of different values in $z \in [0, z_{\mathrm{UB}}]$ is bounded by $\mathcal{O}(\log_{1+\delta} z_{\mathrm{UB}}) = \mathcal{O}(\nu \cdot \log z_{\mathrm{UB}}/\varepsilon)$. Thus, the number of states in the table is polynomial. We conclude that the algorithm runs in polynomial time. ☐

5.2 Speed-scaling for discrete speeds is NP-hard

We show that speed-scaling for discrete speeds is NP-hard. We provide a reduction based on the problem of minimizing the total weighted tardiness of jobs with a common due date, $1|d_j = d|\sum w_j T_j$, which is known to be NP-hard [24]. Here, $T_j = \max\{C_j - d, 0\}$ denotes the tardiness of job j. We use the following generalization of this result for our reduction.

Lemma 21. *The problem of of minimizing $\sum w_j f(C_j)$ on a single machine of unit speed is NP-hard even when f is non-decreasing, piecewise linear and convex with only one breakpoint.*

Proof. Let $\varepsilon \ge 0$ and define the cost function

$$f_\varepsilon(x) = \begin{cases} \varepsilon \cdot x & \text{if } 0 \le x \le d, \\ C_j - d + \varepsilon d & \text{if } d \le x. \end{cases}$$

Note that $T_j = f_0(C_j)$. Now we show that, for $\varepsilon > 0$ small enough, minimizing $\sum_j w_j T_j$ is equivalent to minimizing $\sum_j w_j f_\varepsilon(C_j)$.

Let $k \in \mathbb{N}$, and assume that w_j, p_j and d are natural numbers for all j. It is known that the problem of deciding whether there exists a schedule with $\sum_j w_j T_j \le k$ is NP-hard [24]. Now

notice that

$$\begin{aligned}\sum_j w_j f_\varepsilon(C_j) &= \sum_{j:C_j<d} w_j \varepsilon C_j + \sum_{j:C_j\geq d} (C_j - d + \varepsilon d) w_j \\ &= \varepsilon \cdot \left(\sum_{j:C_j<d} w_j C_j + \sum_{j:C_j\geq d} d w_j \right) + \sum_j w_j f_0(C_j).\end{aligned}$$

Defining $\varepsilon = 1/(d \sum_j w_j) \leq 1$ (which can be described with polynomially many bits) we obtain that

$$\left| \sum_j w_j f_\varepsilon(C_j) - \sum_j w_j f_0(C_j) \right| = \varepsilon \cdot \left(\sum_{j:C_j<d} w_j C_j + \sum_{j:C_j\geq d} d w_j \right) < \varepsilon d \sum_j w_j \leq 1.$$

Therefore $\sum_j w_j f_0(C_j) \leq k$ if and only if $\sum_j w_j f_\varepsilon(C_j) \leq k+1$. We conclude that minimizing $\sum_j w_j f_\varepsilon(C_j)$ is NP-hard, where $\varepsilon \leq 1$ is considered as part of the input. □

Now we are ready to give the main result.

Theorem 22. *The problem of minimizing $\sum_j w_j C_j$ on a single machine for discrete speeds is NP-hard, even if the number of available power levels is* 2.

Proof. The problem with $k > 2$ speed states can be reduced to the case with 2 speed states, by adding dummy states of arbitrarily slow speed. Therefore, we prove hardness of the case of two speeds $s_1 > s_2$.

Consider a scheduling instance on a unit-speed processor with the objective of minimizing $\sum_j w_j f_\varepsilon(C_j)$. We define an equivalent scheduling instance for minimizing $\sum_j w_j C_j$ on a machine with two possible speed states. In the new instance, the job set is the same and the values w_j and v_j for each job j are also preserved. Let $s_1 = 1/\varepsilon$ and $s_2 = 1$. The total energy budget is $E = V + d(1/\varepsilon^{\alpha-1} - 1)$, where V denotes the total work volume, $\sum_j v_j$. A simple interchange argument shows that in an optimal solution the machine runs at decreasing speeds. The time point when the speeds changes is uniquely defined by the energy budget and the total work volume. In this case, the machine runs at speed s_1 until $\tau = \varepsilon d$ and then it runs at speed s_2. It is easy to verify that the total work volume finishes by τ is $\tau \cdot s_1 = d$.

Consider now a schedule without idle time on a machine with the speed profile just described. Assume that by relabeling the jobs the completion times satisfy that $C_1 < C_2 < \ldots < C_n$. Consider scheduling the jobs in a unit speed machine using the same permutation of jobs. In this new schedule the completion times are $C'_j = \sum_{k\leq j} v_k$ for all j. If it easy to check that $f_\varepsilon(C'_j) = C_j$. We conclude that the problem of minimizing $\sum_j w_j f_\varepsilon(C'_j)$ is equivalent to minimizing $\sum_j w_j C_j$ on a machine that has speed $1/\varepsilon$ in interval $[0, \varepsilon d]$ and speed 1 afterwards until all jobs are done. By Theorem 21 both problems are NP-hard, wich concludes the proof. □

5.3 An FPTAS for a constant number of speed states

Let $s_1 > \ldots > s_\kappa \geq 1$. A simple interchange argument shows that an optimal solution chooses the speed non-increasing over time. We construct a schedule in weight-space. There are at most κ jobs that run at more than one speed; call them *split job*.

We may guess the split jobs and they completion weight at an affordable loss in the total cost.

Lemma 23. *At the cost of an increase in the scheduling cost by a factor $1+\mathcal{O}(\varepsilon)$, we may assume that the completion weights of split jobs are integer powers of $1+\beta$ for $\beta = (1+\varepsilon)^{1/(\kappa-1)} - 1$.*

Proof. This follows directly from applying the technique of stretching intervals for each split job; see Sect. 3. Each time, this increases the total cost by a factor $1+\beta$, which amounts to at most a factor $(1+\beta)^{\kappa-1} = 1+\varepsilon$ for at most $\kappa - 1$ split jobs. □

A fixed choice of split jobs and their completion weights partitions the weight-space into κ subintervals $I_1, \ldots, I_\kappa$ with length $|I_i|$, starting at $\omega_i \geq \sum_{\ell=1}^{i-1} |I_\ell|$. To obtain a schedule, we have to fill the remaining jobs non-preemptively in these subintervals (keeping the split jobs where they are). By construction all jobs in one subinterval will run at the same speed, i.e., jobs in weight-interval I_i will run at uniform speed s'_i where $s'_i = s_{n-i+1}$. (Recall that the job sequence in weight space corresponds to the reverse sequence in time space.) We want to find a schedule of minimum total cost *and* we have to control the energy consumption.

5.3.1 Dynamic program

We construct a DP that finds a partition of a set of remaining jobs into κ subsets each of which is assigned to an individual interval I_i. The jobs in each individual set are scheduled according to Smith Rule in weight space, that is, in non-decreasing order of ratios w_j/v_j. Let all jobs be indexed in this order.

The dynamic program generates a state $[k, z, y_1, \ldots, y_{\kappa-1}]$ if there is a feasible schedule of jobs $1, \ldots, k$, in which the total weight scheduled in interval I_i (excluding the split job) is y_i, and where the total value (including split jobs) is $z := \sum_{j=1}^{k} x_j C_j^w$, where $x_j = v_j/s'_i$ for a job j in interval i. The value of the state $[k, z, y_1, \ldots, y_{\kappa-1}]$ is the minimum energy that is necessary for obtaining such a schedule. The dynamic program starts with the states $[0, z, 0, \ldots, 0]$. For each z-value a linear program computes the minimum energy that is necessary to obtain this scheduling value when scheduling only the split jobs J_s. It determines the power that is assigned to each split job and thus their actual execution times. Let ℓ_{ji} be the amount of time that split job j is running at a valid speed s'_i.

$$\begin{aligned}
\min \sum_{j \in J_s} \sum_{i=1}^{\kappa-1} \ell_{ji} P(s'_i) & \\
\sum_{j \in J_s} C_j^w \cdot \sum_{i=1}^{\kappa} \ell_{ji} \leq z & \\
\sum_{i=1}^{\kappa} \ell_{ji} s'_i = v_j \qquad & j \in J_s \\
\ell_{ji} \geq 0, \qquad & j \in J_s, i \in \{1, \ldots, \kappa\} \\
\ell_{ji} = 0, \qquad & j \in J_s,\ s_i \text{ not valid}\,.
\end{aligned}$$

After computing the starting states, the DP computes all states by moving from any state $[j-1, z, y_1, \ldots, y_{\kappa-1}]$ to at most κ new states $[j, z', y'_1, \ldots, y'_{\kappa-1}]$. We distinguish the κ different possibilities of assigning job j: In case that job j is assigned to intervals I_i for some $i \in \{1, \ldots, \kappa - 1\}$ then

$$z' = z + \frac{v_j}{s'_i} \cdot (\omega_i + y_i + w_j) \text{ and } y'_i = y_i + w_j \text{ and } y'_{i'} = y_{i'} \text{ for } i' \neq i\,, \tag{11}$$

provided that $y'_i \leq |I_i| - v_{j_i}$, where j_i is the ith split job. In the case that j is scheduled in I_κ, then

$$z' = z + \frac{v_j}{s'_i} \cdot \left(\omega_\kappa + \sum_{\ell=1}^{j} w_\ell - \sum_{\ell'=1}^{\kappa-1} y_{\ell'} \right) \text{ and } y'_i = y_i,\ i \in \{1, \ldots, \kappa - 1\}\,. \tag{12}$$

In any case, the value of the new state is

$$E[j, z, y_1, \ldots, y_{\kappa-1}] = E[j-1, z, y_1, \ldots, y_{\kappa-1}] + \frac{v_j}{s'_i} \cdot P(s'_i).$$

An optimal schedule can be obtained by finding a state $E[n, z, y_1, \ldots, y_{\kappa-1}] \leq E$ with minimum z and backtracking from that state. Since the z-values are bounded by $Z_{UB} := \sum_{j=1}^{n} w_j (\sum_{\ell=1}^{j} v_\ell / s_\kappa)$ and the y_i-values are bounded by $|I_i|$, the running time of this dynamic programming algorithm is $\mathcal{O}(n \cdot Z_{UB} \cdot \max_i |I_i|^\kappa)$.

5.3.2 FPTAS

In a fully polynomial time algorithm, we can neither afford to consider all possible objective values z, nor can we consider all possible y_i-values. As in the PTAS in Theorem 20, we can reduce the number of possible values of scheduling cost $z \in [0, z_{\mathrm{UB}}]$ to $\mathcal{O}(\nu \cdot \log z_{\mathrm{UB}}/\varepsilon)$ by restricting to powers of $1+\beta'$ with $\beta' = (1+\varepsilon)^{1/\nu} - 1$ and losing a factor $1+\varepsilon$ in the scheduling cost.

The main challenge is to discretize the range of values y_i in an appropriate way. Notice that we cannot afford to round y-values since they contain critical information on how much processing capacity remains in an interval. Perturbing this information causes a considerable change in the set of feasible schedules. In this way one might lose optimal schedules or introduce infeasible schedules with too much processing in an available interval. Both effects cannot be controlled easily, and thus, must be avoided.

The intuition behind the following algorithm is to reduce the number of states by removing those with the same (rounded) objective value and nearly the same total work in available interval I_i. Among them, we want to store those with smallest amount of work in an interval I_i in order to make sure that enough space remains for further jobs that need to be scheduled there.

Algorithm F:

1. For an arbitrary given $\varepsilon > 0$ let $\delta := \varepsilon/n$.
2. Partition each interval I_i into sub-intervals $I_{i,k}$ of length $|I_i|\delta$ for $k = 1, \ldots, \lceil n/\varepsilon \rceil$; the last interval may be smaller, respectively. We say a value y_i *is in interval* $I_{i,k}$ if $\omega_i + y_i \in I_{i,k}$.
3. Run the dynamic program DP with the following modification. Among the states for the same job set and the same (rounded) objective value z, we store at most one for each interval $I_{i,k}$, namely the one with currently minimum y-value within $I_{i,k}$.

Lemma 24. *Suppose that algorithm DP on an instance with n jobs finds a chain of states*[2] $[0,0,0,\ldots,0], [1, z_1^*, y_{1,1}^*, \ldots, y_{\kappa-1,1}^*], \ldots, [n, z_n^*, y_{1,n}^*, \ldots, y_{\kappa-1,n}^*]$. *Then Algorithm F finds for each $j = 1, \ldots, n$ a state $[j, z_j, y_{1,j}, \ldots, y_{\kappa-1,j}]$ with*

$$y_{i,j}^* - j|I_i|\delta \le y_{i,j} \le y_{i,j}^* \qquad \text{and} \qquad z_j \le (1 + (j-1)\delta)\, z_j^* \,. \tag{13}$$

In (13) we state an upper and a lower bound on $y_{i,j}$ in terms of $y_{i,j}^*$. This will turn out to be important when proving the bound on v_j in terms of z_j^* in the following analysis.

Proof. We give a proof by induction on the number of jobs. For one job we store at most two states which obviously fulfill both conditions in (13). Suppose that the lemma is true for j jobs. Now consider state $[j+1, z_{j+1}^*, y_{1,j+1}^*, \ldots, y_{\kappa-1,j+1}^*]$ that was obtained from $[j, z_j^*, y_{1,j}^*, \ldots, y_{\kappa-1,j}^*]$ according to (11) or (12). We distinguish the two cases.

First case: If state $[j+1, z_{j+1}^*, \ldots, y_{\kappa-1,j+1}^*]$ was obtained from a state $[j, z_j^*, y_{1,j}^*, \ldots, y_{\kappa-1,j}^*]$ by adding job $j+1$ to interval I_i, for some $i \in \{1, \ldots, \kappa-1\}$, then Algorithm F when doing the same starting from $[j, z_j, y_{1,j}, \ldots, y_{\kappa-1,j}]$ yields $z_{j+1} = z_j + v_{j+1}/s_i' \cdot (\omega_i + y_j + w_{j+1})$ and $\bar{y}_{i,j+1} = y_{i,j} + w_{j+1}$ while it keeps $\bar{y}_{i',j+1} = y_{i',j}$ for all $i' \neq i$. However, we cannot guarantee that this state will survive, because we might find a partial solution with the same objective value z_{j+1} but smaller values $y_{i',k}$ within the same subinterval $I_{i',k}$, for $i' \in \{1, \ldots, \kappa-1\}$. But in this case $\bar{y}_{i',j+1} - y_{i',j+1}$ is bounded from above by the length of interval $I_{i',k}$ and thus by $|I_{i'}|\delta$. Thus, it holds for all $i' \neq i$:

$$y_{i',j+1}^* \ge y_{i',j+1} \ge y_{i',j} - |I_{i'}|\delta s_\kappa \ge y_{i',j}^* - (j+1)|I_{i'}|\delta \,,$$

and for the i-th y-value we get

$$y_{i,j+1}^* \ge y_{i,j+1} \ge y_{i,j} + w_{j+1} - |I_i|\delta \ge y_{i,j}^* - j|I_i|\delta + w_{j+1} - |I_i|\delta = y_{i,j+1}^* - (j+1)|I_i|\delta.$$

[2] *Chain* of states means that, for $j = 0, \ldots, n-1$, state $[j+1, z_{j+1}^*, y_{1,j+1}^*, \ldots, y_{\kappa-1,j+1}^*]$ is obtained from $[j, z_j^*, y_{1,j}^*, \ldots, y_{\kappa-1,j}^*]$ by adding job $j+1$ according to (11) or (12).

Moreover,

$$z_{j+1} \leq z_j + \frac{v_{j+1}}{s'_i} \cdot (\omega + y_{i,j} + w_{j+1}) \leq (1+(j-1)\delta)\, z_j^* + \frac{v_{j+1}}{s'_i} \cdot (\omega + y_{i,j}^* + w_{j+1})$$
$$\leq (1+j\delta)\, z_{j+1}^* \,.$$

Second case: If state $[j{+}1, z_{j+1}^*, y_{1,j+1}^*, \ldots, y_{\kappa-1,j+1}^*]$ was obtained from state $[j, z_j^*, y_{1,j}^*, \ldots, y_{\kappa-1,j}^*]$ by adding $j+1$ to the last available interval I_κ, then $y_{i,j+1}^* = y_{i,j}^*$ for all $i \in \{1, \ldots, \kappa-1\}$ and Algorithm F yields $\bar{y}_{i,j+1} = y_{i,j}$ when doing the same. If some values $\bar{y}_{i,j+1}$ are later replaced by some smaller values $y_{i,j+1}$ in the same interval $I_{i,k}$, we still get

$$y_{i,j+1}^* \geq y_{i,j+1} \geq y_{i,j} - |I_i|\delta \geq y_{i,j+1}^* - (j+1)|I_i|\delta.$$

Moreover,

$$\begin{aligned}
z_{j+1} &= z_j + \frac{v_{j+1}}{s'_\kappa} \cdot \left(\omega_\kappa + \sum_{\ell=1}^{j+1} w_\ell - \sum_{\ell'=1}^{\kappa-1} y_{\ell',j}\right) \\
&\leq (1+(j-1)\delta)\, z_j^* + \frac{v_{j+1}}{s'_\kappa} \cdot \left(\omega_\kappa + \sum_{\ell=1}^{j+1} w_\ell - \sum_{\ell'=1}^{\kappa-1} (y_{\ell',j}^* - j|I_{\ell'}|\delta)\right) \\
&\leq (1+(j-1)\delta)\, z_j^* + (1+j\delta) \cdot \frac{v_{j+1}}{s'_\kappa} \cdot \left(\omega_\kappa + \sum_{\ell=1}^{j+1} w_\ell - \sum_{\ell'=1}^{\kappa-1} y_{\ell'}^*\right) \\
&\leq \ (1+j\delta)\, z_{j+1}^* \,.
\end{aligned}$$

The inequalities are due to induction hypothesis, $\sum_{\ell'=1}^{\kappa-1} |I_{\ell'}| \leq \omega_\kappa$, and (12). □

Now we can prove the main result.

Theorem 25. *There is an FPTAS for speed-scaling with a given energy budget for* $\min \sum w_j C_j$ *on a single machine with constantly many discrete speeds.*

Proof. Let OPT denote the value of an optimal solution. Lemma 24 guarantees that Algorithm F finds an end state that corresponds to a solution for all jobs of value $z_n \leq (1+(n-1)\delta)z_n^* \leq (1+\varepsilon)z_n^*$, where z_n^* is an optimal solution when restricting to objective values that are multiples of $\varepsilon Z_{LB}/n$. Since the rounding error for at most n jobs accumulates to at most εZ_{LB}, we have that $z_n^* \leq (1+\varepsilon)\mathrm{OPT}$ and thus $z_n \leq (1+\varepsilon)^2\mathrm{OPT}$.

It remains to show that the running time is polynomial in the input and $1/\varepsilon$. When restricting a split job's completion weight to powers of $1+\beta$ for $\beta = (1+\varepsilon)^{1/(\kappa-1)} - 1$ then the number of different completion weights for one split job is $\mathcal{O}(\log_{1+\beta} \sum_{j \in J}) w_j \in \mathcal{O}(\kappa \cdot \nu/\varepsilon)$. Thus, the algorithm considers $\mathcal{O}((\kappa \cdot \nu/\varepsilon)^{\kappa-1})$ many possibilities for split jobs and their completion weight. For each situation, we have $\mathcal{O}(\nu \cdot \log z_{\mathrm{UB}}/\varepsilon) = \mathcal{O}(\nu \cdot n/\varepsilon^2)$ possible objective values. Concerning the y-values, we consider $1/\delta$ subintervals for each interval I_i for $i \in \{1, \ldots, \kappa-1\}$. Since $\delta = \varepsilon/n$ this amounts to $(n/\varepsilon)^{\kappa-1}$ different value assignments for $\kappa-1$ intervals. Thus, the algorithm has running time $\mathcal{O}(n^\kappa \nu^\kappa \kappa^{\kappa-1}/\varepsilon^{2k})$. □

5.4 FPTAS for a constant number of different-speed intervals

In this section we briefly discuss a different application of the technique presented in Section 5.3. Consider for the problem of scheduling jobs on a single machine with a constant number of time intervals each of uniform speed. Applying the ideas from Section 5.3 in *time-space* yields an FPTAS. More precisely, we obtain the following result.

Theorem 26. *There exists an FPTAS for non-preemptive*[1] *scheduling to minimize* $\sum w_j C_j$ *on a single machine with a constant number of intervals of different, but uniform speed. For the resumable*[1] *setting, there is an FPTAS in the same setting when the maximum ratio of speeds is bounded.*

To obtain this result we again guess the split jobs, the availability intervals that they cross (in time-space), and their completion *time*.

Lemma 27. *We may assume that the completion times of split jobs are powers of* $1+\varepsilon/s$ *by loosing at most a factor* $1+\varepsilon$ *in the total scheduling cost.*

Proof. Consider an optimal solution with completion times for (at most) $\kappa - 1$ split jobs $C_1^*, \ldots, C_{\kappa-1}^*$. Now, for $i = 1, \ldots, \kappa - 1$ do the following: round C_i^* up to the nearest power of $C_i := (1+\varepsilon/s)^k$, $k \in \mathbb{N}$, and shift the entire job to match this new completion time. We also adopt the schedule of the subsequent jobs in such a way that there is no idle time created, that is, no job is delayed but at least one job previously completing after i is processed now at least to some extent before i. Then no job but i is delayed and possibly executed at some lower speed. The amount of delay is bounded by

$$\frac{s_i \cdot \left((1+\frac{\varepsilon}{s})^{k+1} - (1+\frac{\varepsilon}{s})^k\right)}{s_\kappa} = \frac{s_i \cdot \varepsilon \cdot (1+\frac{\varepsilon}{s})^k}{s \cdot s_\kappa} \leq \varepsilon \cdot \left(1+\frac{\varepsilon}{s}\right)^k \leq \varepsilon \cdot C_i^* \,.$$

Thus, given an optimal schedule and rounding for each job i the value C_i^* as described may lead to an increase of the completion time C_i by at most $\varepsilon \cdot C_i^*$, that is,

$$C_j' = C_j + \varepsilon \cdot C_i^* \leq (1+\varepsilon) C_j \,.$$

□

With the given speeds in each interval I_i we can compute also the starting time for each split job. This gives us a partition into time intervals in which we have to fill the remaining jobs non-preemptively. To do so, we apply the FPTAS from the speed-scaling setting in a simplified form in time-space. We simply compute all states without minimizing over energy. We omit a repetition of the DP and the analysis.

We note that the above algorithm computes a permutation of jobs and thus a resumable scheduling solution. In a non-preemptive model the guessing step for split jobs is not necessary since there are no split jobs. In that case, we run the FPTAS above at a running time independently of the speed ratios.

6 Preemptive scheduling with release dates

A simple list scheduling approach allows to apply our previous results in the energy setting with non-trivial release dates.

Algorithm List-Scheduling:

1. Compute a solution sequence π for the relaxed input instance with all release dates set to 0.
2. Apply preemptive list scheduling according to π, that is, run at any time t the job j with the highest priority in π among the available ($r_j \leq t$) but uncompleted jobs.

Theorem 28. *Algorithm* List-Scheduling *can be implemented as a factor* $(2+\varepsilon)$ *approximation for continuous and discrete speed-scaling when jobs have individual release dates.*

Proof. In the first step we run the PTAS from Theorem 19 in the continuous-speed setting and from Theorem 20 in the discrete-speed setting, respectively. We keep the energy distribution over jobs and thus their speeds s_j as computed. Then we only need to argue on the increase of scheduling cost by preemptive list scheduling respecting release dates in the second step. The completion time of a job j is $C_j = r_j + \sum_{k=1}^{j} v_k/s_k$. Now, the weighted sum over all jobs is bounded by the lower bound $\sum_j w_j r_r$ and $(1+\varepsilon)$ times the lower bound obtained by solving the relaxed problem without release dates. This concludes the proof. □

[1]Resumable jobs may run during a speed-0 interval; non-preemptive jobs must not.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proc. of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 32–43, 1999.

[2] S. Albers. Energy-efficient algorithms. *Commun. ACM*, 53(5):86–96, 2010.

[3] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3, 2007.

[4] N. Bansal and K. Pruhs. The geometry of scheduling. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 407–414. IEEE Computer Society, 2010.

[5] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM J. Comput.*, 39(4):1294–1308, 2009.

[6] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[7] S.-H. Chan, T.-W. Lam, and L.-K. Lee. Non-clairvoyant speed scaling for weighted flow time. In M. de Berg and U. Meyer, editors, *Algorithms – ESA 2010*, volume 6346 of *LNCS*, pages 23–35. Springer Berlin / Heidelberg, 2010.

[8] M. Cheung and D. Shmoys. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'2011)*, volume 6845 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2011.

[9] F. Diedrich, K. Jansen, U. Schwarz, and D. Trystram. A survey on approximation algorithms for scheduling with machine unavailability. In *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*, pages 50–64. Springer, 2009.

[10] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on a single machine. *SIAM Journal on Computing*, 41:565–586, 2012.

[11] M. X. Goemans and D. P. Williamson. Two-dimensional Gantt charts and a scheduling algorithm of Lawler. *SIAM Journal on Discrete Mathematics*, 13:281–294, 2000.

[12] W. Höhn and T. Jacobs. On the performance of Smith's rule in single-machine scheduling with nonlinear cost. In D. Fernández-Baca, editor, *Proceedings of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*, volume 7256 of *Lecture Notes in Computer Science*, pages 482–493. Springer, 2012.

[13] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[14] I. Kacem and A. Mahjoub. Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 56(4):1708–1712, 2009.

[15] H. Kellerer and V. Strusevich. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica*, 57:769–795, 2010.

[16] C.-Y. Lee. Machine scheduling with availability constraints. In J.-T. Leung, editor, *Handbook of scheduling*. CRC Press, 2004.

[17] Y. Ma, C. Chu, and C. Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58:199–211, 2010.

[18] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994.

[19] K. Pruhs, P. Uthaisombut, and G. J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4, 2008.

[20] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15, 2000.

[21] S. Stiller and A. Wiese. Increasing speed scheduling and flow scheduling. In *Proceedings of the 21st Symposium on Algorithms and Computation (ISAAC 2010)*, volume 6507 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2010.

[22] G. Wang, H. Sun, and C. Chu. Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, 133:183–192, 2005.

[23] F. F. Yao, A. J. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 374–382, 1995.

[24] J. Yuan. The NP-hardness of the single machine common due date weighted tardiness problem. *Systems Science and Mathematical Sciences*, 5(4):328–333, 1992.